Assignment 2

Injamuri Krutika, 18MCMT20

Question 1

Color Half Toning:

Ordered Color dithering is the algorithm that is implemented. The implementation is experimented with various masks, one being the famous Bayers masks and the rest of them are customised masks.

Observations:

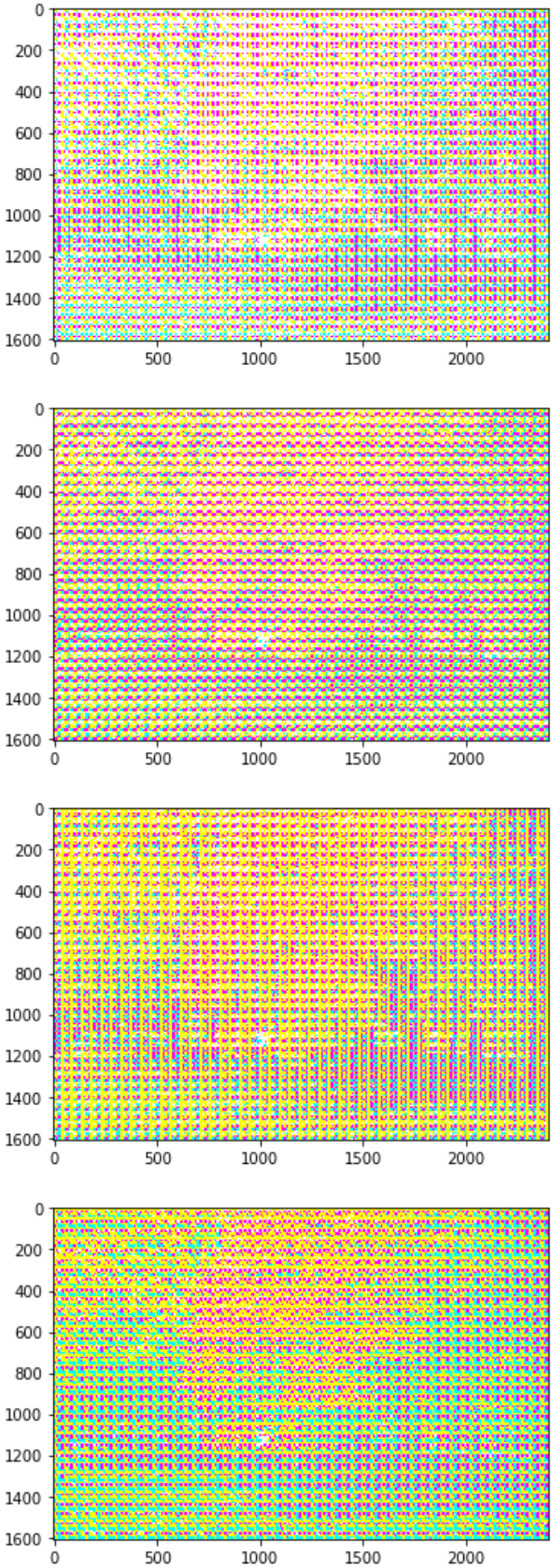
- When the new image taken is filled with white i.e., [255,255,255] the output is very much similar to the original image
- When the new image taken is filled with black i.e., [0,0,0] the output is black tinted and the color of the image is also not similar to the original image.

In [14]:
```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
```

In [15]:
```
  masks = [
# Bayers Mask
( [[2,2],[0,2],[2,0],[3,1],[2,3]] ,
  [[1,1],[1,3],[0,1],[0,3],[2,1],[1,2]],
  [[0,0],[3,3],[1,0],[3,2],[3,0]]),

# Custom Mask  1
( [[0,0],[1,2],[2,1],[3,1],[3,3]] ,
 [[0,2],[0,3],[1,0],[1,1],[2,3],[3,0]],
 [[0,1],[1,3],[2,0],[2,2],[3,2]]),

# Custom Mask  2
( [[0,1],[0,3],[2,1],[2,2],[2,3]] ,
 [[0,2],[1,1],[1,3],[3,1],[3,3],[2,0]],
 [[0,0],[1,0],[1,2],[3,0],[3,2]]),

# Custom Mask  3
([[0,0],[1,1],[3,3],[0,3],[2,1],[3,0]],
[[2,2],[2,0],[1,0],[0,1],[1,2]],
[[0,2],[1,3],[3,1],[2,3],[3,2]])
]
```

In [16]:
```
def cmy_color_dithering(image,result_image_name,masks):
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    new_image = np.full((image.shape[0]*4, image.shape[1]*4, 3), 255,np.uint
8)
    image_extension = 0
    for c_order,m_order,y_order in masks:
        image_extension += 1
        for i in range(image.shape[0]):
            for j in range(image.shape[1]):
                c_val = 255 - image[i,j,0]
                m_val = 255 - image[i,j,1]
                y_val = 255 - image[i,j,2]

                #find number of dots to be set for C,M,Y using unitary metho
d
                c_dots = ( len(c_order) * c_val ) // 255
                m_dots = ( len(m_order) * m_val ) // 255
                y_dots = ( len(y_order) * y_val ) // 255

                x,y = i*4, j*4
                for k in range(c_dots): # for cyan, set B and G
                    new_image[x+c_order[k][0],y+c_order[k][1]] = [0,255,255]

                for k in range(m_dots): # for magenta, set B and R
                    new_image[x+m_order[k][0],y+m_order[k][1]] = [255,0,255]

                for k in range(y_dots): # for yellow, set G and R
                    new_image[x+y_order[k][0],y+y_order[k][1]] = [255,255,0]
        plt.imshow(new_image)
        plt.show()
        cv.imwrite(result_image_name+str(image_extension)+".jpg",cv.cvtColor
(new_image,cv.COLOR_RGB2BGR))
```

In [17]:
```
image = cv.imread('images/fall-colours.jpg')
cmy_color_dithering(image,'images/results/q1.fall-colours_dithered',masks)
```



Question 2

My own Error Diffusion Technique:

In this the error in the single pixel is diffused to the immediate right and the immediate left pixels with a coefficient of 1/2 each.

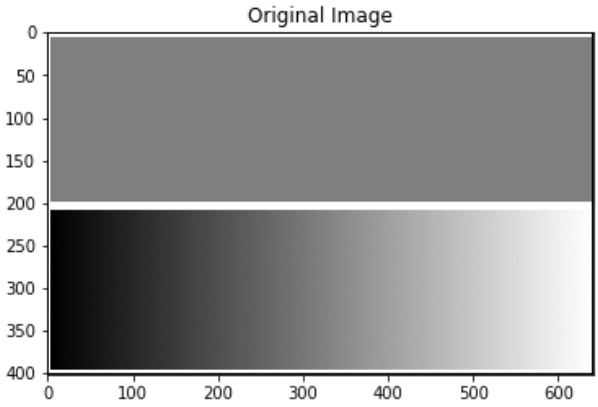
Observations:

- In Floyd Steinbergs algorithm there is a checkerboard sort of pattern created in the first half of the image because it is evenly distributed with 128.
- In my own error diffusion algorithm the checker board pattern has some noise in it.
- In Floyd Steinbergs algorithm the transition in the gray shade is crealy observed.
- The more the number of pixels to which the error is diffused better is the checkerboard pattern.

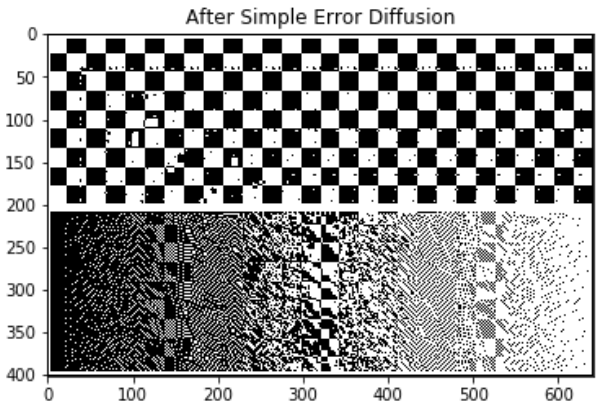
In [18]:
```
def get_final_val(val, factor, error):
    val += factor * error
    val = min(val, 255)
    val = max(0, val)
    return val

def simple_error_diffusion(image):
    height, width, channels  = image.shape[0], image.shape[1], image.shape[2
]
    for i in range(channels):
        for j in range(1, height-1):
            for k in range(1, width-1):
                error = image[j,k,i] - 255 if image[j,k,i] >= 128 else image
[j,k,i]
                image[j,k,i] = 255 if image[j,k,i] >= 128 else 0
                image[j,k+1,i] = get_final_val(image[j,k+1,i], 1/2, error)
                image[j+1,k,i] = get_final_val(image[j+1,k,i], 1/2, error)
    return image
```

In [19]:
```
def floyd_steinberg(image):
    height, width, channels  = image.shape[0], image.shape[1], image.shape[2
]
    for i in range(channels):
        for j in range(1, height-1):
            for k in range(1, width-1):
                error = image[j,k,i] - 255 if image[j,k,i] >= 128 else image
[j,k,i]
                image[j,k,i] = 255 if image[j,k,i] >= 128 else 0
                image[j+1,k,i] = get_final_val(image[j+1,k,i] , 7/16, error)
                image[j-1,k+1,i] = get_final_val(image[j-1,k+1,i] , 3/16, er
ror)
                image[j,k+1,i] = get_final_val(image[j,k+1,i] , 5/16, error)
                image[j+1,k+1,i] = get_final_val(image[j+1,k+1,i] , 1/16, er
ror)
    return image
```

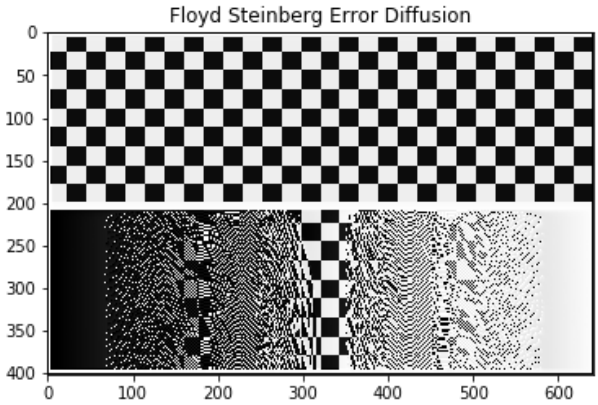
In [20]:
```
image = cv.imread('images/ed-eg.png')
image_rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB)
image_rgb_pad = cv.copyMakeBorder( image_rgb, 1, 1, 1, 1, cv.BORDER_CONSTANT
)
plt.imshow(image_rgb_pad)
plt.title("Original Image")
plt.show()
```



In [21]:
```
image = cv.imread('images/ed-eg.png')
image_rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB)
image_rgb_pad = cv.copyMakeBorder( image_rgb, 1, 1, 1, 1, cv.BORDER_CONSTANT
)
error_diffused_img = simple_error_diffusion(image_rgb_pad)
cv.imwrite("images/results/q2.error_diffused_img.png",error_diffused_img)

plt.imshow(error_diffused_img)
plt.title("After Simple Error Diffusion")
plt.show()
```



In [22]:
```
image = cv.imread('images/ed-eg.png')
image_rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB)
image_rgb_pad = cv.copyMakeBorder( image_rgb, 1, 1, 1, 1, cv.BORDER_CONSTANT
)
floyd_error_diff = floyd_steinberg(image_rgb_pad)
cv.imwrite("images/results/q2.error_diff_floyd_steinberg.png",floyd_error_di
ff)
plt.imshow(floyd_error_diff)
plt.title("Floyd Steinberg Error Diffusion")
plt.show()
```



Question 3

Color to Gray Scale using Color Filter

In [23]:
```
def colour_to_gray (image, color_filter_array):
    height, width, channels  = image.shape[0], image.shape[1], image.shape[2
]
    new_image = np.zeros((height,width))
    for i in range(0, height - 1, 2):
        for j in range(0, width - 1, 2):
            new_image[i  , j  ] = image[i][j][color_filter_array[0][0]]
            new_image[i+1, j  ] = image[i+1][j][color_filter_array[0][1]]
            new_image[i  , j+1] = image[i][j+1][color_filter_array[1][0]]
            new_image[i+1, j+1] = image[i+1][j+1][color_filter_array[1][1]]
    return new_image
```

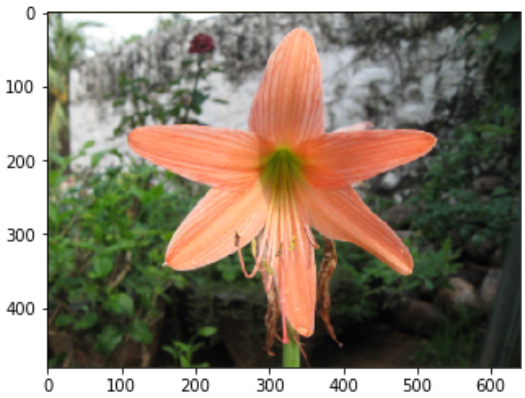
In [24]:
```
image = cv.imread('images/orange-flower.ppm')
image_rgb = cv.cvtColor(image, cv.COLOR_BGR2RGB)
image_rgb_pad = cv.copyMakeBorder( image_rgb, 1, 1, 1, 1, cv.BORDER_CONSTANT
)
color_filter_array = np.array([[2,1],[1,0]]) #BGGR
gray_image = colour_to_gray(image_rgb_pad, color_filter_array)

cv.imwrite("images/results/q3.colour_to_gray.png",gray_image)
```

Out[24]: True

Gray Scale Image to Color Image

In [25]:
```
def interpolate(img_pad, isG=False):
    new_image = img_pad.copy()
    g_conv_matrix = np.array([[0, 1, 0],
         [1, 4, 1],
         [0, 1, 0]]) / 4

    rb_conv_matrix = np.array([[1, 2, 1],
     [2, 4, 2],
     [1, 2, 1]]) / 4

    (height, width) = img_pad.shape
    window = [0] * 9
    for i in range(1, height-1):
        for j in range(1, width-1):
            window[0] = img_pad[i-1, j-1]
            window[1] = img_pad[i-1, j]
            window[2] = img_pad[i-1, j+1]
            window[3] = img_pad[i, j-1]
            window[4] = img_pad[i, j]
            window[5] = img_pad[i, j+1]
            window[6] = img_pad[i+1, j-1]
            window[7] = img_pad[i+1, j]
            window[8] = img_pad[i+1, j+1]
            window = np.array(window).reshape(3, 3)
            if isG is True:
                new_image[i,j] = (window * g_conv_matrix).sum()
            else:
                new_image[i,j] = (window * rb_conv_matrix).sum()
            window = window.flatten()

    return new_image

def demosaic(gray_image, _filter):
    height, width, channels  = image.shape[0], image.shape[1], image.shape[2
]
    new_image = np.zeros((height,width,3))
    for i in range(0, height - 1, 2):
        for j in range(0, width - 1, 2):
            new_image[i  , j  ][_filter[0][0]] = gray_image[i][j]
            new_image[i+1, j  ][_filter[0][1]] = gray_image[i+1][j]
            new_image[i  , j+1][_filter[1][0]] = gray_image[i][j+1]
            new_image[i+1, j+1][_filter[1][1]] = gray_image[i+1][j+1]
    new_image = new_image.astype('uint8')
    r, g, b = new_image[:,:,0], new_image[:,:,1], new_image[:,:,2]
    return np.stack((interpolate(r),
                         interpolate(g, True),
                         interpolate(b)), axis=2)
```

In [28]:
```
color_filter_array = np.array([[2,1],[1,0]]) #BGGR
color_img = demosaic(gray_image.astype('uint8'), color_filter_array)
plt.imshow(color_img)
plt.show()
cv.imwrite("images/results/q3.demosaic.png", cv.cvtColor(color_img, cv.COLOR
_BGR2RGB))
```



Out[28]: True